A Software Approach to Improving Cloud Computing Datacenter Energy Efficiency and Enhancing Security through Botnet Detection

Razvan-Ioan Dinita, *MIEEE*, Adrian Winckles, George Wilson
Computing and Technology Department
Anglia Ruskin University
Cambridge, United Kingdom
{razvan.dinita, adrian.winckles, george.wilson}@anglia.ac.uk

***Abstract*–This work presents positive experiment results on the efficiency and security potential of an optimized and novel approach to an Autonomous Management Distributed System (AMDS) running in a Cloud Computing environment. The results validate the AMDS software design and demonstrate its potential as an industrial application to be used in modern datacenters. On one hand, from an operational performance point of view, they show the AMDS' ability of reconfiguring itself on the fly, thus resulting in 14 percent increased efficiency over the lifetime of the first experiment. On the other hand, they show an overall malicious (Botnet) data packet detection rate of over 52 percent, a significant percentage for only 5000 network data samples analyzed by the Botnet software module plugged into the AMDS. Both experiments have been performed in a VMWare run cloud environment, however due to the AMDS' abstract architecture, it has the potential to interface with any existing cloud management system that exposes an API.**

I. INTRODUCTION

Computing is a term referring to goal-oriented activities that revolve around algorithmic processes [1]. Ever since the middle of last century, the scientific computing domain has evolved more rapidly than any other technology. It began with highly simplistic computing devices, quickly moving to more complex devices, and, for the better part of last half of the 20th century, evolving into very large clusters of interconnected computing devices working towards the same pre-set computational goals, almost identical to utility computing [2].

"Cloud Computing" is a higher-level term used to describe distributed computing, which involves data flows across real-time, high-speed networks. This enables the possibility of running intensely computational applications across multiple physical devices in parallel. This term, when used by laypeople, generally refers to generally available online services. These services, although appearing to be supported by physical hardware, are in fact running on virtualized hardware simulated by software operating on real machines. One of the main advantages of this approach is the ability to scale, up or down, these virtual infrastructures on the fly without any noticeable service disruptions, similar to a cloud [3].

This work started with the authors critically developing a novel method of optimizing cloud networks in terms of energy consumption and system operation. The novelty consists of the software's ability to reconfigure itself on the fly based on live network readings [4]. The authors also critically developed a novel method to prevent, detect, and stop network intrusions and malicious behavior in a cloud environment [5].

Finally, the work presents flexible software solution (as opposed to the general approach of using rigid hardware) to a general communications/networking problem. The software solution automatically acquires data about network traffic and hardware loads of a cloud infrastructure, analyzes them and redistributes loads for efficient energy management and optimal data communication parameters (security, data transfer speed, access wait times, power consumption) [5], [6].

II. BACKGROUND

The research presented in [7] discusses a software-based approach to enhancing network security within Cyber Physical Systems (CPSs). They propose a software abstraction implementation called Bundle that allows network administrators to program a wide range of different applications within 60 lines of code or less involving multiple CPSs aimed at reducing complexity when working with CPSs as well as maintain acceptable levels of energy consumption across the network.

A different approach is presented by [8]. It proposes a full-stack Cloud Management solution called Snooze. This would be directly in charge of all hardware and virtual resources available within the cloud infrastructure it manages.

Overall this is a very capable solution, building on work done by many other similar solutions [9]-[11], utilising all of their strengths and disregarding many of their drawbacks, however it requires to be deployed by itself on a cloud infrastructure, rather than act as a plugin to an already established infrastructure, like in the case of the work presented by the authors of this paper.

III. AMDS Core, Performance Experiment – Setup, Results, and Discussion

The scope of this experiment is focused on AMDS performance measurements through self produced logs detailing each action and connection undertaken by the AMDS, together with the time in milliseconds each task has taken to complete.

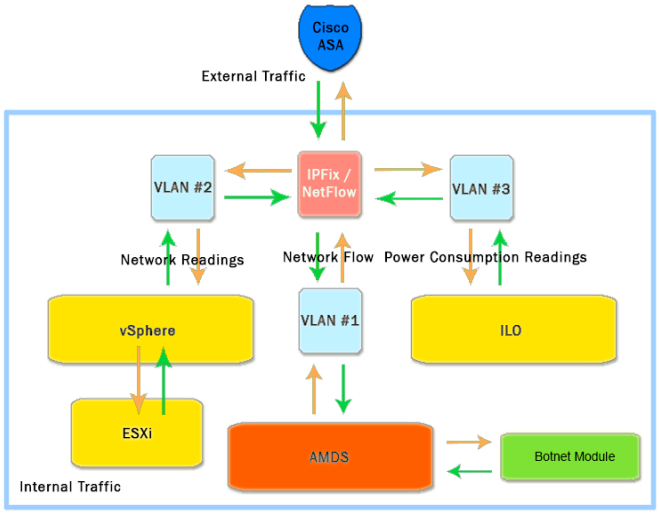

Fig. 3 - AMDS Experiment Logical Layout - AMDS Performance Measurements

As can be seen in Fig. 3, the setup also has the AMDS Botnet Module activated, which analyses NetFlow/IPFix network samples. This helps accurately determine the AMDS production performance on a live datacenter infrastructure.

A. *Experiment Setup*

By default, the AMDS records all actions undertaken during its lifetime, including their start and end time, employing (1) to calculate each action's total running time. These contain information on what triggered each action, the evaluated data, the result of the evaluation and the command issued to the action initiator, such as vSphere, an ILO, a switch, or a router.

$$T_{total} = T_{end} - T_{start}. \quad (1)$$

After several log data sets have been analyzed, a performance report is generated that provides an overall view on the AMDS operational timings.

B. *Experiment Results*

The data presented in the below Table 1 is an extract from the AMDS operational log produced throughout the AMDS lifetime. In order to remain within this experiment's boundaries, several irrelevant columns have been stripped out. The data represent several steps undertaken by two AMDS modules, the Botnet and the Control modules respectively, when processing both external and internal information.

TABLE 1
AMDS Operational Log Extract

Action by	Operation	Started at T_{start}	Ended at T_{end}	Total (ms) T_{total}
AMDS_botnet	networkFlow_retrieve	05/08/2014 13:06:11.433	05/08/2014 13:06:11.541	108
AMDS_botnet	networkFlow_storeRaw	05/08/2014 13:06:11.541	05/08/2014 13:06:11.585	44
AMDS_botnet	networkFlow_breakdown	05/08/2014 13:06:11.585	05/08/2014 13:06:12.011	26
AMDS_botnet	networkFlow_analyse	05/08/2014 13:06:12.011	05/08/2014 13:06:16.270	4259
AMDS_botnet	networkFlow_storeAnalysis	05/08/2014 13:06:16.270	05/08/2014 13:06:16.319	49
…….	…….	……	……	……
AMDS_control	efficiency_analyse	05/08/2014 13:07:12.262	05/08/2014 13:07:14.183	1921
AMDS_control	efficiency_storeAnalysis	05/08/2014 13:07:14.183	05/08/2014 13:07:14.245	62
AMDS_control	vsphere_sendCommand	05/08/2014 13:07:14.245	05/08/2014 13:07:14.258	13
……	…….	……	……	……

The results presented in Table 2 are average execution times of each of the more important AMDS modules: Botnet, Control, Auth, and Conn modules, respectively. These latency timings have been observed over approximately 3 hours of uninterrupted AMDS operational lifetime.

TABLE 2
AMDS Average Module Operation Times (ms)

	Botnet Module (ms)	Control Module (ms)
Initial readings	4486	2145
End of Experiment Average Time	3855	2117
Change vs. Initial readings (%)	-14%	-1.3%
Period of time Observed (approx.)	05/08/2014 13:00 – 05/08/2014 16:00	

The "Initial Readings" table row represents the very first time each of the observed modules has completed one task. The "End of Experiment Average Times" table row have been calculated as averages of the time it has taken each module to start and complete subsequent tasks, until the end of the experiment.

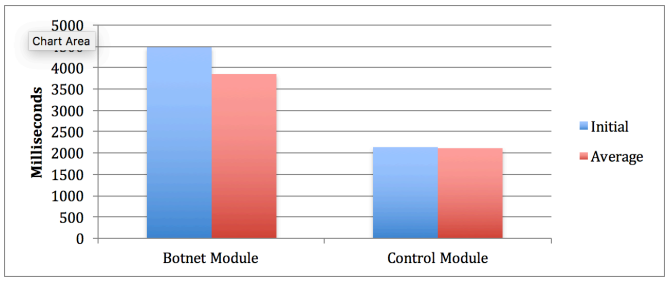

Fig. 4 - Performance Experiment Results Latency Comparison

As can be seen in Fig. 4, the Botnet module average latency has declined by approximately 14% during the lifetime of this experiment, caused by the fact that with each new network flow the user activity model improves, and as such it is able to determine the flow threat level slightly faster each time. This is a significant latency reduction, which can only improve over time as the user activity model becomes better refined.

Also in Fig. 4, the Control module average latency has also declined by approximately 1.3%. The most likely cause of this is the different latency expressed by the network devices it interfaces with, which are influenced mostly by external traffic fluctuations.

C. *Discussion*

The findings of this experiment were positive, with the two main components that were tested, the Control and Botnet modules, operating well within expected parameters. The Control module achieved an initial operation latency average of 2,145 milliseconds, dropping to approximately 2,117 milliseconds by the end of the experiment, while the Botnet module achieved an initial operation latency average of 4,486 milliseconds, dropping to approximately 3,855 milliseconds. The results reflect the design and implementation approach taken by the authors in developing the AMDS, which have allowed it to reduce its operational latency times by up to 14% during its lifetime through self-reconfiguration. Due to its asynchronous nature, this has virtually no impact on day-to-day datacenter operations, while at the same time having a positive impact on the energy management and security aspects of the infrastructure.

IV. AMDS Botnet Module, Detection Experiment – Setup, Results, and Discussion

The scope of this experiment is focused on testing the AMDS Botnet Module's detection capabilities through a bigger, more standardized set of infected network packets.

A. *Experiment Setup*

In addition to the existing setup seen in Fig. 3, the authors have also added an external Botnet-like attacker using the botnet code written by Charles Leifer[1]. This is a simple program that allows a master machine to control other, infected machines through IRC commands. This experiment assumes said machines are already infected.

As can be seen in Fig. 5, a new logical node has been created and made available to the existing infrastructure by interfacing with the ASA (Adaptive Security Appliance) and becoming part or regular network traffic. This new logical node is a very simple Botnet application, consisting of a master and several workers. This allows for infected data

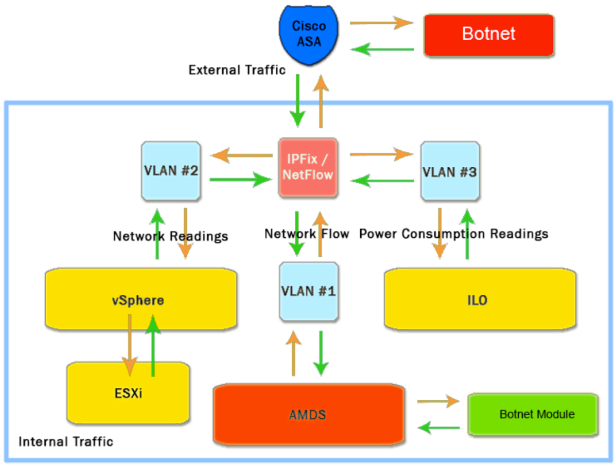

Fig. 5 - AMDS Experiment Logical Layout - AMDS Botnet Module Detection Capabilities

packets to be randomly introduced into the system alongside regular, healthy data packets.

The packets used in this experiment vary in size, but are typically between 100 and 500 bytes. Regular user traffic is typically around the 500 byte mark, which should give the existing model ample leeway to adapt.

B. *Experiment Results*

The experiment has run over an extensive period of time and it has produced a large dataset. An overview of this data, along with some of the experiment parameters, can be seen in Table 3.

TABLE 3
Data Packet Analysis Results

	Sample #500	Sample #1500	Sample #3000	Sample #5000
# Packets (1000s)	181	189	207	247
~ Packet Size (Bytes)	552	529	483	405
# Infected (1000s)	0	0	85	128
# Detected (1000s)	0	0	29	67
Detection Rate (%)	0	0	34.1	52.3

For the first half of the experiment, as can be seen in Table 3, regular packets with an average size of 529-552 bytes have been filtered through the AMDS Botnet Detection Module. This has been used as a training mechanism for the heuristic algorithm discussed in [5], so it would later on have a healthy packet model to compare infected packets against.

[1] https://github.com/coleifer/irc/tree/master/botnet

Fig. 6 - Data Packet Distribution per 10% Sample

As can be seen in Fig. 6, using the 1 Gbps network link has evaluated into approximately 1.81 million data packets, of which 181 thousand of the 500 initial readings for analysis have been sampled.

For the second half of the experiment, a random percentage of Botnet generated data packets have been introduced through the use of actual botnet code alongside the regular data packets used in the first half of the experiment. This had a direct impact on the data packet size as this has decreased the average packet size of the samples by as much as 24%, from 529 to 405 bytes each.

Starting with packet #3000, the heuristic algorithm attempts to detect malicious packets by comparing potentially infected packets to the healthy network traffic model it has built prior to the introduction of infected packets.

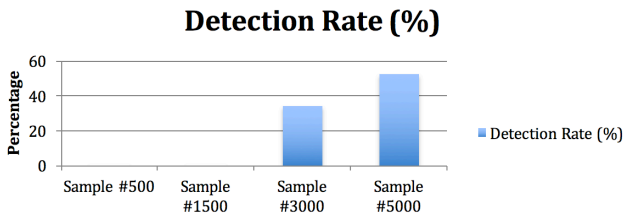

Fig. 8 - Botnet Packet Detection Rate

As can be seen in Fig. 8, the AMDS, through the use of its Botnet Detection heuristic algorithm, has managed to detect approximately 34.1% of all infected packets at the start of the Botnet attack. This detection rate has steadily increased up until the end of the experiment to approximately 52.3% of all infected packets.

C. *Discussion*

The results presented above give a clear indication of the potential of the AMDS having its Botnet Detection Module activated. Applying the heuristic algorithm to more and more data packet samples allows the AMDS Botnet Detection module to better understand what real world Botnet data packets look like, and detect more similar packets or even unknown Botnet packet types in the future.

V. CONCLUSIONS

This paper has presented experimental results of an Autonomous Management Distributed System (AMDS) operating within a VMWare-run Datacenter.

The experiments were intended to show the operational efficiency of the AMDS as well as the benefits it brings from a security point of view in terms of performing Botnet detection based on network flow analysis.

As discussed in section IV of this paper, the malicious activity detection success rate has grown steadily over time up to 52.3% of all infected network communications. The main driver of this increase over time is the system's ability to reconfigure itself on the fly based on changes happening within and around the virtual environment it resides in, novel design features presented in [4]. These results demonstrate what can be achieved using a purely software approach to an existing datacenter hardware infrastructure.

These results reinforce the potential of this software design as a datacenter enhancement and security system, with virtually no impact on day-to-day operations due to the asynchronous nature of the implementation.

VI. FUTURE WORK

The authors intend to expand the Botnet module to include more detection algorithms to help with analyzing network traffic, as well as gain other monitoring capabilities, such as accessing individual active Virtual Machines and looking at running processes and processor loads and even comparing them to a set of VM profiles (e.g. for a VM only used for static web pages, high processor load may indicate unusual activities).

REFERENCES

[1] Joint Task Force for Computing Curricula (JTFCC), "Computing Curricula 2005 – The Overview Report," *ACM*, 2005, ISBN 1-59593-359-X

[2] M. Carroll, P. Kotzé, A. Van Der Merwe, "Securing virtual and cloud environments," *Cloud Computing and Services Science*, pp. 73-90, Springer New York, 2012

[3] CORDIS "Providing a platform for a coordinated response to cloud cybercrime," 2013, Available at: http://cordis.europa.eu/news/rcn/36249_en.html

[4] R. I. Dinita, G. Wilson, A. Winckles, M. Cirstea, T. Rowsell, "A Novel Autonomous Management Distributed System for Cloud Computing Environments," *Industrial Electronics Conference (IECON), 2013 39th Annual Conference of*, November 2013

[5] R. I. Dinita, A. Winckles, G. Wilson, "Use of NetFlow/IPFIX Botnet Detection Tools to Determine Placement for Autonomous VMs," *Cybercrime Forensics Education and Training (CFET), 2014 7th International Conference on*, July 2014, ISBN 97801909067158

[6] R. I. Dinita, G. Wilson, A. Winckles, M. Cirstea, A. Jones, "Hardware Loads and Power Consumption in Cloud Computing Environments,", *International Conference on Industrial Technology (ICIT)*, pp. 1291-1296, February 2013, ISBN 978-1-4673-4568-2

[7] P. A. Vicaire, E. Hoque, Z. Xie, J. A. Stankovic, "Bundle: A Group-Based Programming Abstraction for Cyber-Physical Systems," *Industrial Informatics, IEEE Transactions on*, pp. 379-392, May 2012, doi: 10.1109/TII.2011.2166772

[8] E. Feller, M. Simonin, Y. J´egou, A.-C. Orgerie, D. Margery, "Snooze: A Scalable and Autonomic Cloud Management System," *[Research Report] RR-8649*, Inria Rennes, pp. 31, 2014

[9] "OpenStack," Available at: https://www.openstack.org

[10] "OpenNebula," Available at: http://opennebula.org

[11] "Nimbus," Available at: http://www.nimbusproject.org